Converting Adiabatic Equations into Quantum Gate Circuits

Anushka Gupta - agupta35
Sreeraj Rajendran - srajend2
Varun Garg - vsgarg

3rd December 2018

1 Introduction

Our project aims to translate QUBOs (Quadratic Unconstrained Binary Optimization) problems into quantum gate circuits. The current quantum computing architectures for adiabatic computing support far more qubits than the gate model. However, the gate model is more versatile and is expected to be faster than classical computing. Hence, having the ability to translate adiabatic quantum equations to gate circuits is beneficial in the long run as eventually the gate model will be better than adiabatic quantum computers.

2 Implementation

2.1 Algorithm

The code basically accepts a QUBO for a boolean problem. It takes 2 (n,m) inputs initially (number of input bits and number of output bits). The code assumes that the first n bits are the input bits and the latter m bits are the output bits. It takes the weights of all the qubits and then the strengths of the couplers.

1. Receive input number and output number of qubits. The output bits are always the bits in the end.
2. Brute force all possible states in the QUBO.
3. Sort the states as per the energy outputs of the QUBO.
4. Build circuit for the lowest energy states.

The proposed method outputs a quantum gate circuit corresponding to the inputs of the lowest energy states in the QUBO. Since, we are working with boolean QUBOs, the circuit are created using NOT, CNOT and Toffoli gates. The circuits are reversible and therefore are according to the postulates of quantum mechanics.

2.2 Gates to be implemented

We try to work with boolean logical circuits like AND, OR, NAND, NOR, half adder and full adders. The Hamiltonian for the gates are as below:

1. **AND Gate:**

   $0 \cdot a_1 + 0 \cdot a_2 + 2 \cdot a_3 + 1 \cdot b_{12} - 1.5 \cdot b_{13} - 1.5 \cdot b_{23}$

2. **OR Gate:**

   $1 \cdot a_1 + 1 \cdot a_2 + 1 \cdot a_3 + 1 \cdot b_{12} - 2 \cdot b_{13} - 2 \cdot b_{23}$

3. **NAND Gate:**

   $-1 \cdot a_1 - 1 \cdot a_2 - 1.5 \cdot a_3 + 0.5 \cdot b_{12} + 1 \cdot b_{13} + 1 \cdot b_{23}$

4. **NOR Gate:**

   $-1 \cdot a_1 - 1 \cdot a_2 - 1 \cdot a_3 + 1 \cdot b_{12} + 2 \cdot b_{13} + 2 \cdot b_{23}$

5. **XOR:**

   $1 \cdot a_1 + 1 \cdot a_2 + 2 \cdot a_3 + 4 \cdot a_4 + 3 \cdot b_{12} - 3 \cdot b_{13} - 4.5 \cdot b_{14} - 3 \cdot b_{23} - 4.5 \cdot b_{24} + 5 \cdot b_{34}$

6. **Half Adder:**

   $1 \cdot a_1 + 1 \cdot a_2 + 1 \cdot a_3 + 4 \cdot a_4 + 2 \cdot b_{12} - 2 \cdot b_{13} - 4 \cdot b_{14} - 2 \cdot b_{23} - 4 \cdot b_{24} + 4 \cdot b_{34}$

7. **Full Adder:**

   $1 \cdot a_1 + 1 \cdot a_2 + 4 \cdot a_3 + 8 \cdot a_4 + 1 \cdot a_5 + 2 \cdot b_{12} + 4 \cdot b_{13} - 6 \cdot b_{14} - 2 \cdot b_{15} + 4 \cdot b_{23} - 6 \cdot b_{24} - 2 \cdot b_{25} - 11 \cdot b_{34} - 5 \cdot b_{35} + 7 \cdot b_{45}$

3 How to run

1. Run the provided python code (qc_generalized.py) in an environment with pandas, qiskit and matplotlib.
2. It will ask for the number of qubits and their weights and strengths in the QUBO.
3. The final qiskit code will be created in the directory from which the initial python code was run and it is called qiskitProgram.py.
4. To run the circuit, execute python3 qiskitProgram.py.
5. Give the input values for the qubits, the circuit will execute on the local simulator for 1024 times and give the counts of all of the measured outputs.

4 Sample output

```
// xor shown
 python3 qc_generalized.py
Enter number of input qubits: 2
Enter number of output qubits: 2
Enter weight 1: 1
Enter weight 2: 1
Enter weight 3: 2
Enter weight 4: 4
Enter strength 1 - 2: 3
Enter strength 1 - 3: -3
Enter strength 1 - 4: -4.5
Enter strength 2 - 3: -3
Enter strength 2 - 4: -4.5
Enter strength 3 - 4: 5
[[0, 0, 0, 0, 0.0], [0, 0, 0, 1, 4.0], [0, 0, 1, 0, 2.0], [0, 0, 1, 1, 11.0],
[0, 1, 0, 0, 1.0], [0, 1, 0, 1, 0.5], [0, 1, 1, 0, 0.0], [0, 1, 1, 1, 4.5],
[1, 0, 0, 0, 1.0], [1, 0, 0, 1, 0.5], [1, 0, 1, 0, 0.0], [1, 0, 1, 1, 4.5],
[1, 1, 0, 0, 5.0], [1, 1, 0, 1, 0.0], [1, 1, 1, 0, 1.0], [1, 1, 1, 1, 1.0]]
Minimum values
[0, 0, 0, 0, 0.0]
[0, 1, 1, 0, 0.0]
[1, 0, 1, 0, 0.0]
[1, 1, 0, 1, 0.0]
```

4.1 Qiskit Program genereated for XOR

```
  import numpy as np
import matplotlib.pyplot as plt
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from qiskit import available_backends, execute
from qiskit.tools.visualization import matplotlib_circuit_drawer as drawer, qx_color_scheme
from qiskit.tools.visualization import circuit_drawer

qx = QuantumRegister(2)
cx = ClassicalRegister(2)
qz = QuantumRegister(2)
cz = ClassicalRegister(2)
circuit = QuantumCircuit(qx,qz,cx,cz)

a=np.zeros((2))
for i in range(2):
a[i]=int(input("Enter input bit " + str(i) + " : " ))
if a[i] == 1:
circuit.x(qx[i])

circuit.cx(qx[1], qz[0])
circuit.cx(qx[0], qz[0])
circuit.ccx(qx[0],qx[1],qz[1])


circuit.measure(qz,cz)
circuit.measure(qx,cx)

print(circuit.qasm())

# Execute

print("Running on simulator...")
backend = 'qasm_simulator'
job = execute(circuit, backend, shots=1024)
result = job.result()
print(result.get_counts(circuit))

circuit_drawer(circuit)
```

5 Circuit Results

1. **AND:**

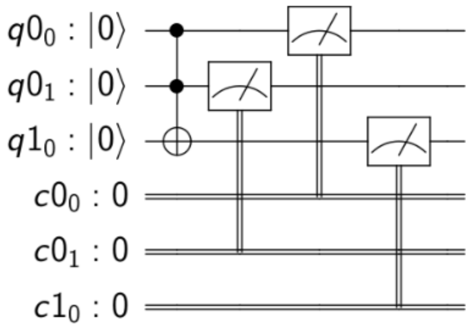

Figure 1: Generated AND circuit

2. **OR:**

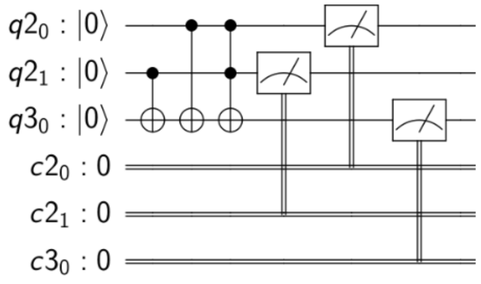

Figure 2: Generated OR circuit

3. **NAND:**

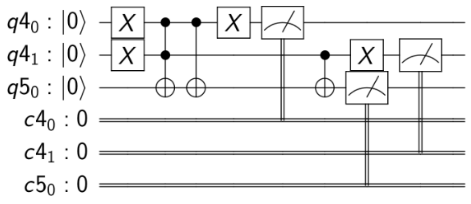

Figure 3: Generated NAND circuit

4. **NOR:**

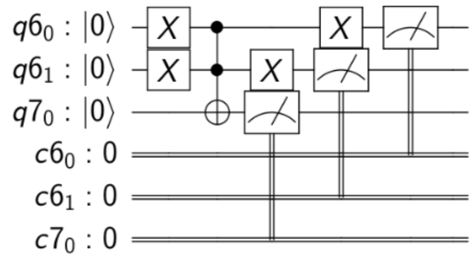

Figure 4: Generated NOR circuit

5. **XOR:**

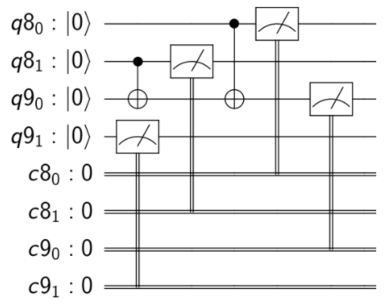

Figure 5: Generated XOR circuit

6. **Half Adder:**

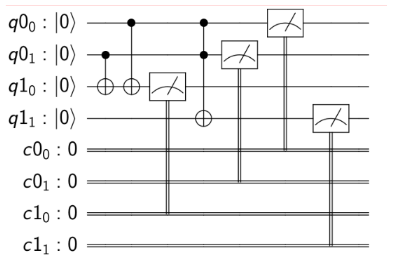

Figure 6: Generated half adder circuit

7. **Full Adder:**

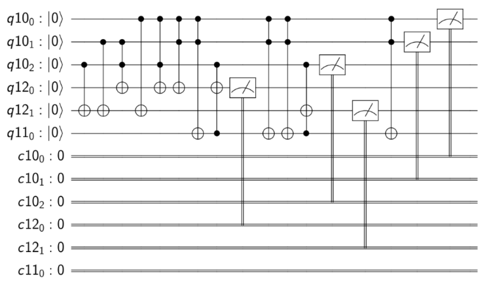

Figure 7: Generated Full adder circuit

5.1 Error in Full adder

In the above circuit, input 111 corresponds to an output of 10 which actually should be a 11. This is an error because of the algorithm we have used. The algorithm applies ccnots based on the input qubits having value 1 and the output qubit having value 1. Since, 111 gives 11 the ccnots are applied twice and therefore 10 is obtained as the output. All other input states give the right answers.

6 Conclusion/Future scope

Many boolean adiabatic equations can now be converted in quantum circuits. The number of gates used for the equations are equal to the circuits created by hand. The only exception is full-adder where an extra set of ccnots are added. These extra gates are the main cause of error.

The converter only works with boolean circuits and does not consider the possibility of superposition. Quantum supremacy relies on the concept of superposition. In the future, superposition may be defined via a QUBO and the converter has to be tweaked to work with Hadamard gates and rotations.

The converter only outputs Qiskit code. There are other Quantum architectures that aren't supported, such as Ion-Trap. Adding extensions to support those architectures is also a component of the future scope of this project.

7 Appendix

The report is present at the given link. Project Webpage

Github Repo with all the files. Git Repository

References

[1] Richard H. Warren. Gates for Adiabatic Quantum Computing Avalaible at:https://arxiv.org/ftp/arxiv/papers/1405/1405.2354.pdf